Informe de Trabajo Final

Elección del framework

Para la realización de este trabajo se utilizó el framework Laravel, en su versión con soporte a largo plazo (LTS) más actual al momento de comenzar el desarrollo, la 5.5.
Para decidirnos por este framework se tuvieron en cuenta los siguientes puntos:

1. Es un framework PHP que utiliza la arquitectura MVC.
2. Posee una documentación muy extensa y abarcativa.
3. Es el framework PHP más utilizado hoy en día[1], lo que implica la existencia de una mayor comunidad de usuarios y foros más activos para realizar y encontrar consultas, entre otras cosas.

Referencias

Como guías a través del desarrollo principalmente nos valimos de la documentación oficial de Laravel, que cuenta además con guías que abarcan la gran mayoría de los conceptos de una aplicación MVC y como estos se utilizan en Laravel.
Además, contamos con Laracast, una serie de cursos en video con soporte oficial de Laravel que abarcan gran parte de los conceptos de este framework, además de servir como tutoriales para muchos de los aspectos principales del desarrollo de una aplicación utilizando Laravel.

Aspectos técnicos evaluados

Seguridad y ruteo

Un proyecto Laravel cuenta con la carpeta 'routes', que posee varios archivos, entre ellos 'web.php', utilizado para definir las rutas que utilizará la aplicación web, y 'api.php', utilizado para definir las rutas de una API.
Dentro de cada uno de estos archivos importamos la clase Route, que cuenta con un método de clase por cada operación HTTP, get(), post(), put() y delete(), que reciben como parámetros:

- Una ruta correspondiente a una ruta de nuestra aplicación que queramos definir en forma de string, y se pueden indicar parámetros a través de llaves.
    - Por ejemplo ‘users/{user}’ *‘matcheara’* con cualquier ruta de la forma ‘users/’ seguida por cualquier valor.
- Una función que será ejecutada cuando se *‘matchee’* con la ruta definida utilizando la operación correspondiente al método de la clase route utilizado.
    - También es posible, en lugar de pasar una función como parámetro, pasar un string de la forma *‘Controller@function’*, que ejecutará la función *function* en

[1] https://www.optisolbusiness.com/insight/whats-happening-with-php-mvc-framework

el controlador *Controller*.
Más adelante se especificará cómo se definen los controladores.

Cada ruta, además, permite la llamada a un método de instancia *middleware*[2], el cual recibe como parámetro el middleware a ejecutar. Un middleware es un mecanismo para filtrar y agregar condiciones a peticiones http que se reciban; al recibir tal petición se ejecuta el middleware asociado (si es que lo posee), antes de ejecutar el controlador indicado, este decidirá si se puede responder la petición, en caso contrario se redirecciona a otro recurso. En nuestra aplicación se utilizan varios middlewares para controlar o bien las acciones de los usuarios o bien el estado actual del sitio:

- Autenticación: El middleware *Authenticate* (previamente definido en Laravel), verifica que exista un usuario logueado. Se utiliza por ejemplo para mostrar la página de reportes.
- Autorización: El middleware *Permission* verifica que el usuario actualmente logueado posea autorización para realizar la acción solicitada. El permiso requerido se envía como parámetro al middleware y se especifica como parte de la utilización de este en la declaración de la ruta. Si el usuario no posee el permiso, el requerimiento no se procesa.
- Sitio en mantenimiento: El middleware *SitioEnMantenimiento* verifica que el sitio este activo, al solicitar las rutas que lo requieran, sino muestra una página que informa del actual estado de mantenimiento del sitio.
- Usuario Bloqueado: El middleware *UsuarioBloqueado* verifica que el usuario que esté intentando loguearse este activo, de lo contrario se le informa y la sesión no es iniciada.

El manejo de la autorización de los usuarios a través de roles y permisos se implementa mediante la librería *Spatie*, la cual es desarrollada por terceros, es de código libre y gratuita e implementa soluciones simplificadas para el manejo de roles y permisos en usuarios para el framework laravel.
Los problemas de seguridad vistos durante la cursada son resueltos con herramientas del propio framework. La posibilidad de Cross-site scripting (XSS) queda resuelta por el motor de plantilla que utilizamos, Blade (se profundiza sobre este en la sección de View en implementación del MVC en Laravel), que realiza la verificación en el momento de la renderización. La Inyección de SQL es resuelta por el ORM de Laravel, llamado Eloquent (se profundiza sobre este en la sección del Modelo en implementación del MVC en Laravel); este utiliza PDO en su funcionamiento, evitando así este problema.

Implementación del MVC en Laravel

- Modelo
  - Laravel utiliza Eloquent[3] como su ORM.
  - Los archivos del modelo se encuentran en *'/app'*, estos son clases php que extienden de la clase *Model*, provista por Eloquent.
  - Cada clase del modelo se corresponde con una tabla de la base de datos, por defecto con la tabla que tenga el mismo nombre que la clase pero en minúscula y con una 's' al final (es decir la clase Paciente se corresponderá

[2] https://laravel.com/docs/5.5/middleware
[3] https://laravel.com/docs/5.5/eloquent

con la tabla pacientes), y tendrá variables que se corresponderán con cada uno de los registros de la tabla correspondiente.
    - Estas clases además brindan métodos para realizar consultas a sus respectivas tablas, utilizando sintaxis php en lugar de SQL, garantizando que no habrá inyección de SQL.
        - Métodos de clase, como *'get'*, que devuelve una colección de instancias correspondientes a todos los registros de la tabla.
        - Métodos de instancia, como *'save'*, que realizan un insert a la tabla de un registro con los valores de las variables de la instancia.
- Vistas
    - Se encuentran en la carpeta '/resources/views', nosotros organizamos este directorio a su vez en subcarpetas.
    - Utiliza el motor de plantillas blade[4], que incluye todas las funciones estándar de un motor de plantillas, como herencia, iteraciones, manejo de variables, entre otras.
    - El motor de plantillas Blade, posee métodos aplicables automáticamente en la renderización con el objetivo de evitar los posibles problemas de seguridad de este tipo, tales como XSS.
    - Tienen la extensión *'*.blade.php'*.
    - Pueden ser accedidas desde los controladores a través de la función '*view'*, que recibe como primer parámetro el nombre del archivo (en caso de estar dentro de una carpeta se utiliza *'nombreDeCarpeta.nombreDeArchivo'*), y como segundo parámetro un arreglo nombrado con las variables que se quieren pasar a la plantilla.
- Controladores
    - Se encuentran en la carpeta *'app/Http/Controllers'*, y son clases php que extienden de la clase Controller provista por Laravel.
    - Por estándar, se define un controlador por recurso (que deberá estar representado en un modelo) que incluye las siguientes funciones:
        - index: devuelve una vista con el listado de los recursos.
        - create: devuelve una vista con el formulario de creación del recurso.
        - store: recibe un objeto representando una HTTP request[5], y lo guarda en la base de datos.
        - show: recibe una instancia del recurso representando al registro a mostrar como parámetro y lo muestra.
        - edit: devuelve una vista con el formulario de edición del recurso.
        - update: recibe una instancia del recurso representando al registro a actualizar, y una HTTP request con los datos nuevos, y actualiza el registro en la base de datos
        - destroy: recibe una instancia del recurso representando al registro a eliminar y debe eliminar ese registro.
    - Cuando se definen los nombres estándar para el modelo, el controlador y las rutas, Laravel a través de un middleware garantiza que a los métodos del controlador lleguen las correctas instancias del modelo, por ejemplo, si se

[4] https://laravel.com/docs/5.5/blade
[5] https://laravel.com/docs/5.5/requests

utiliza la ruta *'/pacientes/show/1'* y se la asocia la función show del controlador de pacientes, esta función recibirá una instancia de paciente correspondiente al registro que posea el id 1, sin necesidad de hacer explícitamente la consulta en la base de datos.
    - Además se cuenta con la clase Validator[6] provista por Laravel, que permite fácilmente validar todos los campos de una HTTP request utilizando las distintas reglas[7] que brinda.

Mecanismo para operaciones CRUD

Considerando lo desarrollado en la sección anterior se puede ver que Laravel brinda muchos mecanismos para facilitar las operaciones de CRUD.
Tomemos como ejemplo un recurso 'pacientes' en nuestra base de datos, para implementar un CRUD sobre ese recurso deberíamos:

- Crear una clase Paciente que extienda de la clase *Model*.
- Crear las vistas de los formularios de creación, edición, listado y muestra de un recurso. En general primero se define un template para cada uno de estos y utilizando *blade* estos simplemente heredaron de los ya definidos.
- Crear una clase PacienteController que extienda de la clase model, que utilice la clase del modelo y las vistas definidas anteriormente y defina las funciones estándar ya mencionadas en el apartado anterior.
- En el archivo de rutas, agregar las rutas correspondientes a los métodos definidos en el controlador.

Todo esto se puede realizar de forma muy sencilla a través de artisan, un intérprete de comandos de Laravel sobre el cual ahondaremos en la sección de Migración del trabajo de cursada al FrameInstalación y desarrollowork.

Migración del trabajo de cursada al Framework

Instalación y desarrollo

Para el desarrollo del proyecto, se utilizó Laravel Homestead[8]. Esto funciona utilizando Vagrant[9], una herramienta que permite administrar entornos de máquinas virtuales, y Laravel Homestead viene en forma de Vagrant Box (formato de paquetes de entornos de Vagrant), y te provee de un entorno de programación aislado, que corre Ubuntu 16.04 y dispone de todos los paquetes necesarios para crear un proyecto de Laravel y más, como pueden ser PHP, git, composer, Apache, MySQL, etc.
Gracias a esto, ninguno de los integrantes del grupo debió instalar nada más que vagrant y no nos encontramos con ningún conflicto relacionado a las versiones que utilizara cada uno.

---

[6] https://laravel.com/docs/5.5/validation
[7] https://laravel.com/docs/5.5/validation#available-validation-rules
[8] https://laravel.com/docs/5.5/homestead
[9] https://www.vagrantup.com/

Migración del trabajo de cursada

A la hora de migrar el trabajo de la cursada al framework, se decidió comenzar desde cero el proyecto, y se reutilizo muy poco código del del trabajo desarrollado durante la cursada, ya que un nuevo proyecto de Laravel viene con una estructura predefinida de archivos y carpetas que en gran medida se debe respetar para que este funciona, y que respeta las ubicaciones ya descritas en la sección acerca de cómo implementar MVC, y además cuenta con artisan[10], un intérprete de comandos que permite, entre otras cosas, generar código, lo que nos resultó más fácil de utilizar que tratar de adaptar el código anterior.
Por ejemplo, los pasos listados en la sección *'Mecanismo para operaciones CRUD'*, se pueden realizar utilizando artisan, a través de los siguientes comandos:

```
php artisan make:model Paciente
php artisan make:controller PacienteController --resource
--model=Paciente
```

El primer comando creará la clase del modelo Paciente, que por defecto referencia la tabla pacientes, y el segundo el controlador con los métodos estándar ya definidos, que interactúa con la clase del modelo Paciente, y además las respectivas rutas que invocan a cada uno de los métodos ya definidos. Las vistas correspondientes deben ser creadas manualmente.
Otro punto importante a la hora de migrar el trabajo fue el de la base de datos, durante el trabajo de cursada, habíamos utilizado un script SQL que generaba la base de datos y le cargaba la información inicial. Este script lo manteníamos en el repositorio, y ante una modificación, todos los miembros del grupo debíamos eliminar nuestra versión de la base de datos y ejecutar el script de nuevo.
Laravel nos da herramientas para realizar esto de manera más fácil, a través de *migrations*[11] y *seeders*[12]. Las primeras se encuentran en el directorio *'/database/migrations'* y permiten definir la base de datos a través de sintaxis php, y las segundas, en el directorio *'/database/seeds'* y permiten la carga de la base de datos. Luego con el comando `php artisan migrate --seed` se ejecutan las migrations y seeders definidas. También nos facilita una herramienta de testing llamada factories, que permite crear registros en las tablas llenandolos con datos ficticios. Para esto utiliza la libreria *facker* que genera datos al azar de un tipo y con una característica pedida. La definición de un factory para un modelo determinado se establece en un archivo *.php*, dentro de la carpeta *'/database/factories'*. La creación de los registros, pudiendo especificar una cantidad, se define en el *seeder* que corresponda al modelo.
Al momento de definir la base de datos, no se mantuvieron de forma exacta los nombres definidos por la cátedra para adecuarla a los estándares de Laravel y facilitar la definiciones de modelos.

[10] https://laravel.com/docs/5.5/artisan
[11] https://laravel.com/docs/5.5/migrations
[12] https://laravel.com/docs/5.5/seeding

Conclusiones

La utilización del framework hizo realmente mucho más fácil el desarrollo de la aplicación, brindando soluciones a muchos de los problemas que atravesamos a lo largo de la cursada y cubriendo muchos aspectos del desarrollo de una aplicación MVC que durante la cursada debimos abarcar manualmente. Si bien en la primera parte del desarrollo se debió dedicar una parte importante del tiempo a instalar y configurar el framework, conocer algunos de sus aspectos y conceptos fundamentales y experimentar con este, luego recuperamos ese tiempo durante el desarrollo, debido a lo ágil que resultó.

Para problemáticas donde Laravel no proveía una solución clara, la utilización de librerías de terceros específicas para este framework, aportaron una solución de fácil utilización y aprendizaje.

Creemos que fue de gran utilidad haber desarrollado una aplicación MVC previamente sin utilizar el framework, ya que gracias a esto muchos de los aspectos de Laravel y problemáticas que solucionaba nos resultaban familiares. Además de reconocer gran parte de la estructura de la aplicación y para que debia ser utilizada, y haberla pensado anteriormente con una estructura similar.